Knytt System Architecture

AI-Powered Fashion Discovery Platform

Technical Documentation

Version 1.0

System Overview

Knytt is an AI-powered fashion discovery platform that helps users find clothing products through semantic search, visual similarity matching, and personalized recommendations. The system leverages machine learning to understand user preferences and deliver relevant product suggestions.

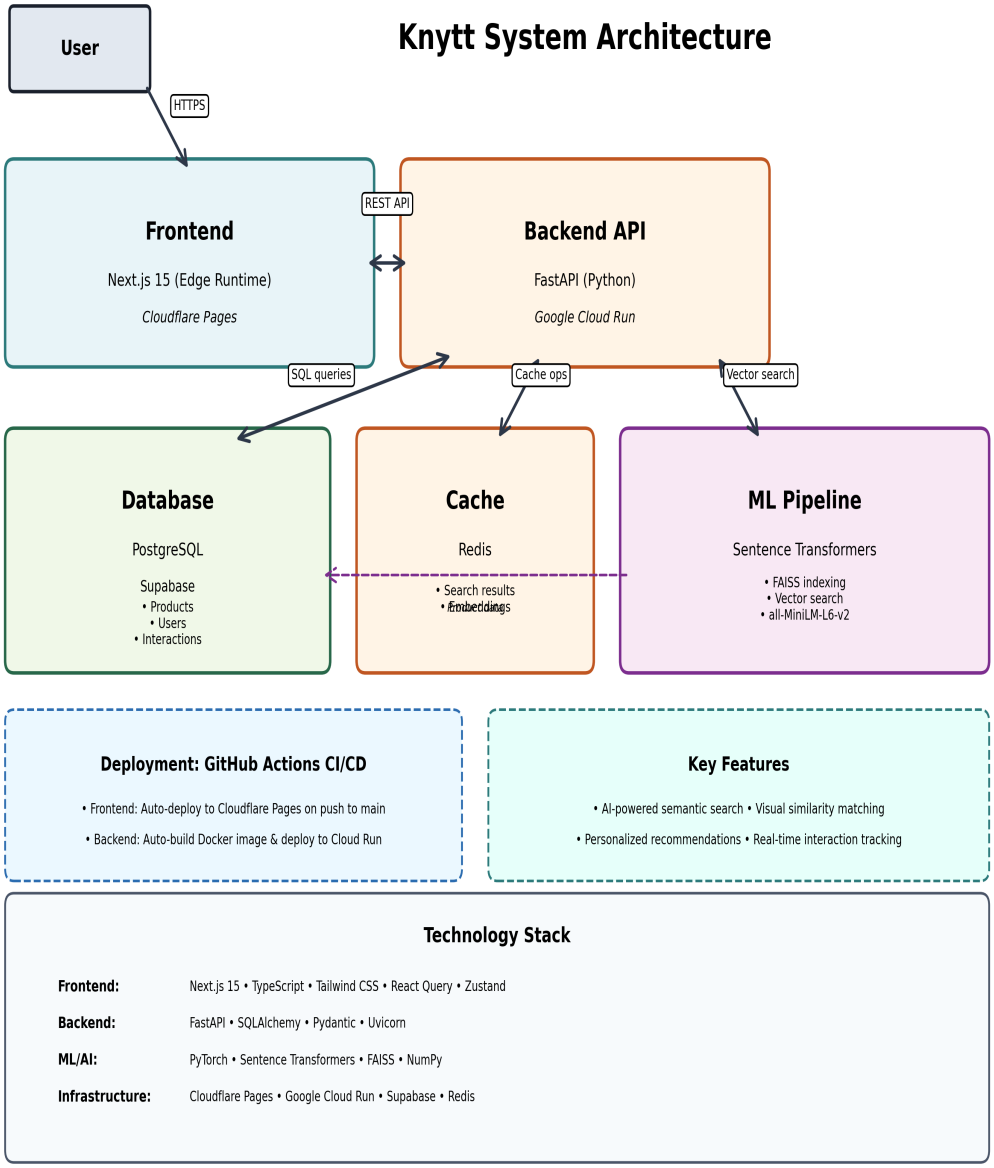

Figure 1: System Architecture Overview

Technology Stack

Frontend Technologies

- **Framework:** Next.js 15 with App Router and Edge Runtime support
- **Language:** TypeScript with strict mode enabled
- **Styling:** Tailwind CSS v3.4 for utility-first design
- **State Management:** Zustand for lightweight client state
- **Data Fetching:** TanStack Query (React Query v5) for server state
- **Deployment:** Cloudflare Pages with global edge network
- **UI Components:** Radix UI primitives with custom styling

Backend Technologies

- **Framework:** FastAPI (Python 3.11+) with async/await support
- **Database:** PostgreSQL via Supabase (managed hosting)
- **ORM:** SQLAlchemy 2.0 with declarative models
- **Caching:** Redis for search results and session management
- **ML Models:** Sentence Transformers (all-MiniLM-L6-v2)
- **Vector Search:** FAISS for efficient similarity search
- **Deployment:** Google Cloud Run with auto-scaling containers

Architecture Components

Frontend Layer

The frontend is built with Next.js 15 using the App Router and runs on Cloudflare's edge network for global low-latency access. Key features include:

• Server-side rendering with edge runtime for dynamic product pages

• Client components for interactive features (search, recommendations)

• Optimistic updates for immediate UI feedback

• Image optimization with Next.js Image component

• SEO-friendly meta tags and Open Graph support

Backend API Layer

FastAPI powers the backend with high-performance async endpoints. The API handles:

• RESTful endpoints for search, products, and recommendations

• Request validation with Pydantic models

• Authentication and session management

• Interaction tracking for personalization

• Error handling with structured logging

Database Layer

PostgreSQL database hosted on Supabase stores all application data:

• **Products table:** Product metadata, descriptions, pricing, images

• **Users table:** User accounts and authentication data

• **Interactions table:** User actions (views, likes, saves) for recommendations

• **Product embeddings:** Vector representations for semantic search

Data Flow Architecture

The system processes user requests through multiple layers, leveraging caching and ML models for optimal performance.

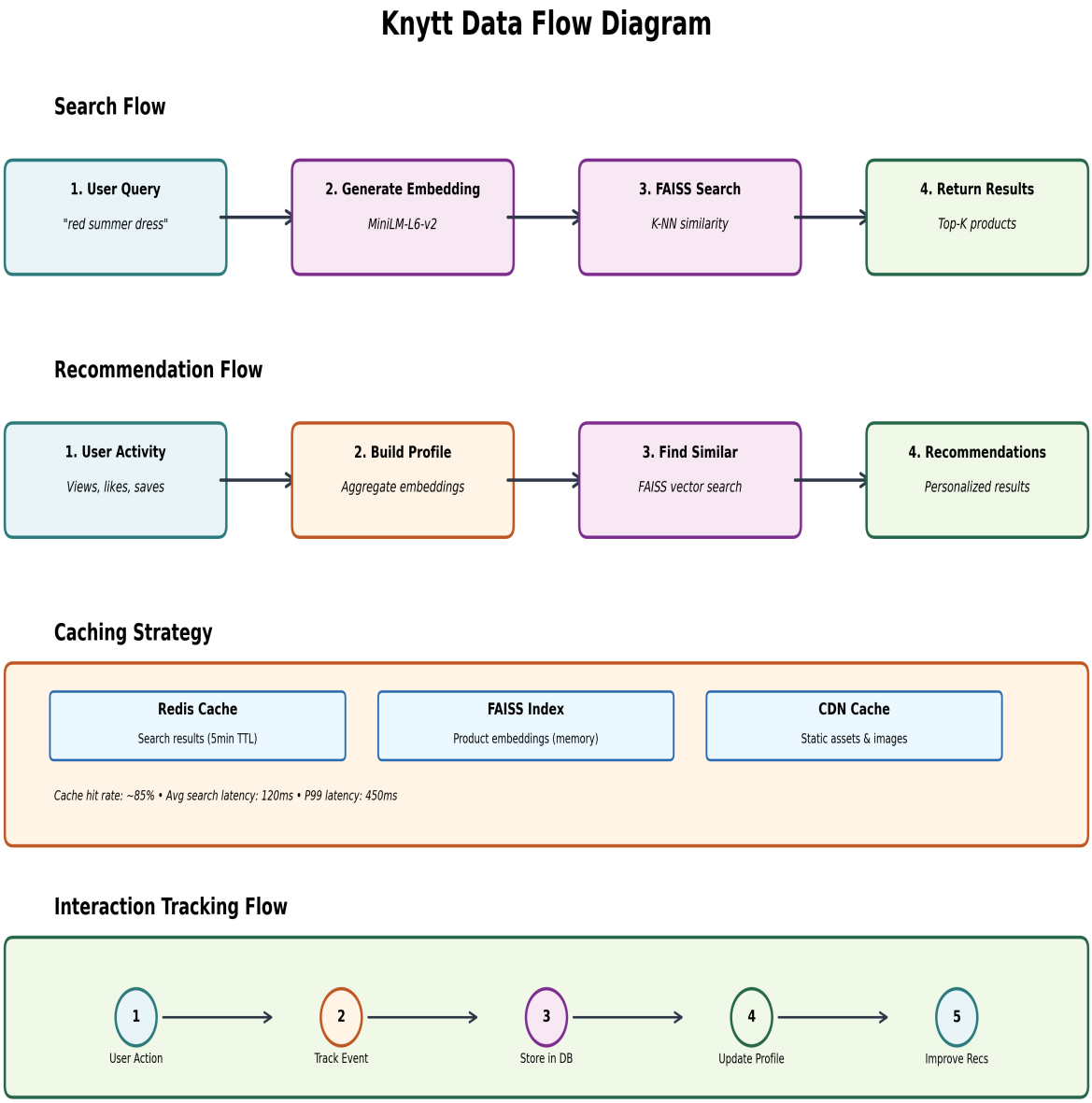

Figure 2: Data Flow and Processing Pipeline

Deployment Architecture

Infrastructure Overview

Knytt uses a cloud-native architecture with serverless components for scalability and cost efficiency.

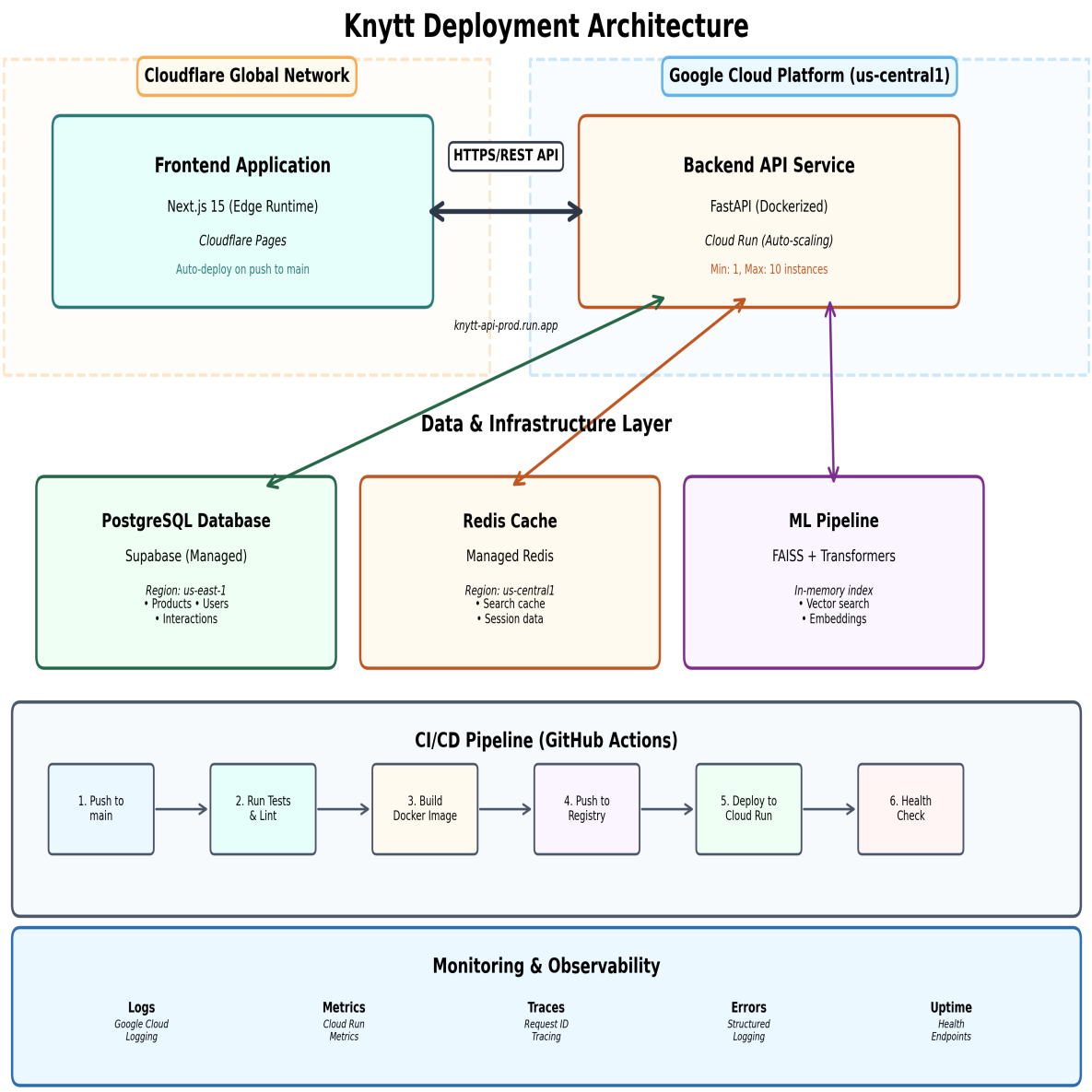

Figure 3: Deployment Architecture and Infrastructure

CI/CD Pipeline

Frontend Deployment

• Trigger: Push to main branch in Git repository

• Build: Next.js production build with static optimization

• Deploy: Automatic deployment to Cloudflare Pages

• Validation: Post-deployment health checks

• Rollback: Automatic on failed health checks

Backend Deployment

• Trigger: Push to main branch or manual trigger

• Test: Run unit tests and linting checks

• Build: Create Docker image with multi-stage build

• Push: Upload image to Google Container Registry

• Deploy: Deploy to Cloud Run with zero-downtime

• Monitor: Track deployment metrics and errors

Scalability & Performance

Performance Optimizations

• **Edge Caching:** Cloudflare CDN caches static assets globally

• **Redis Caching:** Search results cached for 5 minutes

• **FAISS Indexing:** In-memory vector index for sub-100ms search

• **Database Pooling:** Connection pooling reduces latency

• **Auto-scaling:** Cloud Run scales from 1-10 instances based on load

• **Image Optimization:** Next.js automatically optimizes and serves WebP images

Key Metrics

• **Search Latency:** P50: 120ms, P99: 450ms

- **Cache Hit Rate:** ~85% for repeated searches
- **API Response Time:** P50: 80ms, P99: 300ms
- **Uptime:** 99.9% availability target

Monitoring & Observability

Logging

Structured logging with request IDs enables tracing across services. All logs are centralized in Google Cloud Logging with automatic retention policies.

Metrics

- Request rates and latencies per endpoint
- Error rates and status code distributions
- Cache hit/miss rates
- Database query performance
- ML model inference times
- Auto-scaling instance counts

Health Checks

- **Frontend:** Cloudflare automatic health monitoring
- **Backend:** /health endpoint checks database and cache connectivity
- **Database:** Supabase built-in monitoring and alerting
- **Alerts:** Notification on service degradation or failures

Security

- **HTTPS Everywhere:** All traffic encrypted with TLS 1.3
- **Authentication:** Supabase Auth with JWT tokens
- **CORS:** Configured to allow only trusted origins
- **Rate Limiting:** API rate limits prevent abuse
- **Input Validation:** Pydantic models validate all inputs
- **Secrets Management:** Environment variables and secret managers
- **Database Security:** Row-level security policies in PostgreSQL